

1. 多线程的底层实现?

1> 首先搞清楚什么是线程、什么是多线程

2> Mach是第一个以多线程方式处理任务的系统，因此多线程的底层实现机制是基于Mach的线程

3> 开发中很少用Mach级的线程，因为Mach级的线程没有提供多线程的基本特征，线程之间是独立的

4> 开发中实现多线程的方案

- C语言的POSIX接口: `#include <pthread.h>`
- OC的NSThread
- C语言的GCD接口（性能最好，代码更精简）
- OC的NSOperation和NSOperationQueue（基于GCD）

2. 线程间怎么通信?

1> `performSelector:onThread:withObject:waitUntilDone:`

2> NSMachPort

（基本机制：A线程（父线程）创建NSMachPort对象，并加入A线程的run loop。

当创建B线程（辅助线程）时，将创建的NSMachPort对象传递到主体入口点，B线程（辅助线程）就可以使用相同的端口对象将消息传回A线程（父线程）。

http://mobile.51cto.com/hot-403083_all.htm)

3. 网络图片处理问题中怎么解决一个相同的网络地址重复请求的问题?

利用字典（图片地址为key，下载操作为value）

4. 用NSOperation和NSOperationQueue处理A,B,C三个线程,要求执行完A,B后才能执行C,怎么做?

// 创建队列

```
NSOperationQueue *queue = [[NSOperationQueue alloc] init];
```

// 创建3个操作

```
NSOperation *a = [NSBlockOperation blockOperationWithBlock:^(  
    NSLog(@"operation1---");  
)];
```

```
NSOperation *b = [NSBlockOperation blockOperationWithBlock:^(  
    NSLog(@"operation1---");  
)];
```

```
NSOperation *c = [NSBlockOperation blockOperationWithBlock:^(  
    NSLog(@"operation1---");  
)];
```

// 添加依赖

```
[c addDependency:a];
```

```
[c addDependency:b];
```

```
// 执行操作
```

```
[queue addOperation:a];  
[queue addOperation:b];  
[queue addOperation:c];
```

5. 列举cocoa中常见对几种多线程的实现，并谈谈多线程安全的几种解决办法及多线程安全怎么控制？

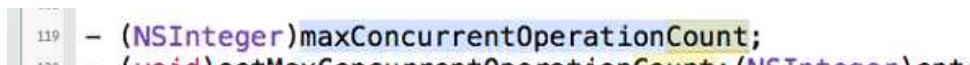
- 1> 只在主线程刷新访问UI
- 2> 如果要防止资源抢夺，得用synchronized进行加锁保护
- 3> 如果异步操作要保证线程安全等问题，尽量使用GCD(有些函数默认就是安全的)

6. GCD内部怎么实现的

- 1> iOS和OS X的核心是XNU内核，GCD是基于XNU内核实现的
- 2> GCD的API全部在libdispatch库中
- 3> GCD的底层实现主要有Dispatch Queue和Dispatch Source
 - Dispatch Queue：管理block(操作)
 - Dispatch Source：处理事件(MACH端口发送,MACH端口接收,检测与进程相关事件等10种事件)

7. 你用过NSOperationQueue么？如果用过或者了解的话，你为什么要使用NSOperationQueue，实现了什么？请描述它和GCD的区别和类似的地方（提示：可以从两者的实现机制和适用范围来描述）。

- 1> GCD是纯C语言的API，NSOperationQueue是基于GCD的OC版本封装
 - 2> GCD只支持FIFO的队列，NSOperationQueue可以很方便地调整执行顺序、设置最大并发数量(FIFO 就是先进先出)
- NSOperationQueue设置最大并发数量的方法



```
119 - (NSInteger)maxConcurrentOperationCount;
```

- 3> NSOperationQueue可以在轻松在Operation间设置依赖关系，而GCD需要写很多的代码才能实现
- 4> NSOperationQueue支持KVO(键值观察者)，可以监测operation是否正在执行(isExecuted)、是否结束(isFinished)，是否取消(isCancelled)
- 5> GCD的执行速度比NSOperationQueue(封装GCD,更高层的东西,性能不好(因为还要转换成GCD).快

如何选择两个:

任务之间不太互相依赖: GCD

任务之间有依赖\或者要监听任务的执行情况: NSOperationQueue

8. 既然提到GCD, 那么问一下在使用GCD以及block时要注意些什么? 它们两是一回事儿么? block在ARC中和传统的MRC中的行为和用法有没有什么区别, 需要注意些什么?

Block的使用注意:

1. block的内存管理(注意循环引用,默认在栈中(不需要内存管理),通过copy就在堆中,就要注意内存管理)
2. 防止循环retain
 - 非ARC (MRC) : __block
 - ARC: __weak__unsafe_unretained

9. 在异步线程中下载很多图片,如果失败了,该如何处理?请结合RunLoop来谈谈解决方案.(提示:在异步线程中启动一个RunLoop重新发送网络请求,下载图片)

1> 重新下载图片

2> 下载完毕, 利用RunLoop的输入源回到主线程刷新UIImageView

10. Socket的实现原理及Socket之间是如何通信的

1).Socket: 称之为套接字,是一种用于网络传输的”工具”.

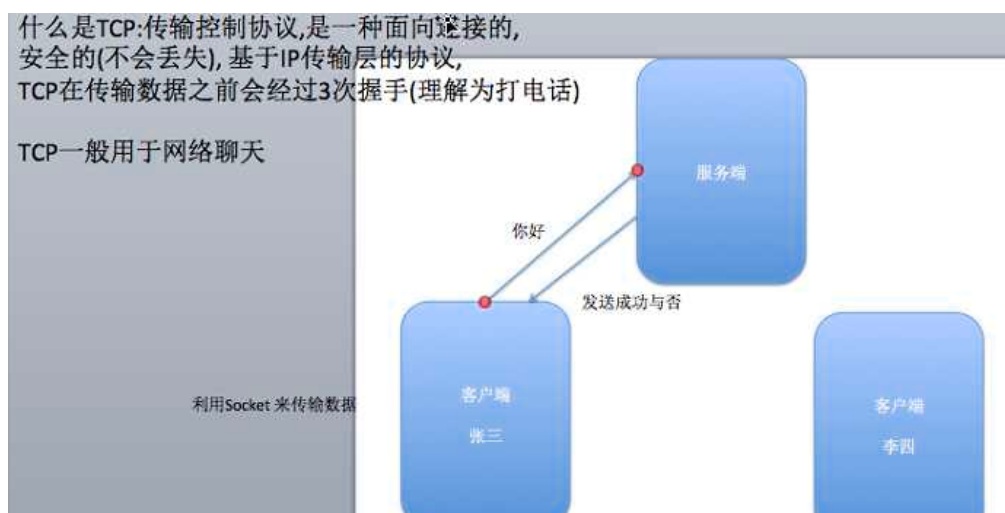
必须告诉ta要传输什么内容,Socket才能传输

网络传输涉及的两个东西(客户端和服务端,之间相互发送信息)

2).Socket实现原理:是基于TCP/UDP的.

加分回答:

3).TCP:传输控制协议,是一种面向连接的,安全的(一般情况下不会丢失的),基于IP传输层的协议.TCP在传输数据之前会经过3次握手(我们可以理解为打电话,嘟嘟,



滴的一声——喂有人么——得到回应)长连接,客户端可以主动给服务端发送请求,服务端也可以主动给客户端发送请求,聊天要保证数据的安全性.

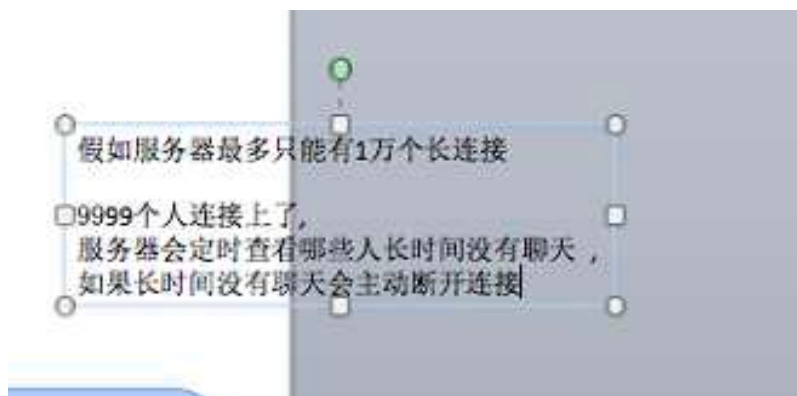
后面的XMPP基于TCP.是老外写的一个即使通信的框架.

特点:慢 相对于UDP来说

UDP:传输控制协议,是一种面向连接的,不安全的(可能会丢失数据),基于IP传输层的协议. 特点:快(只管发,不管收到没有)

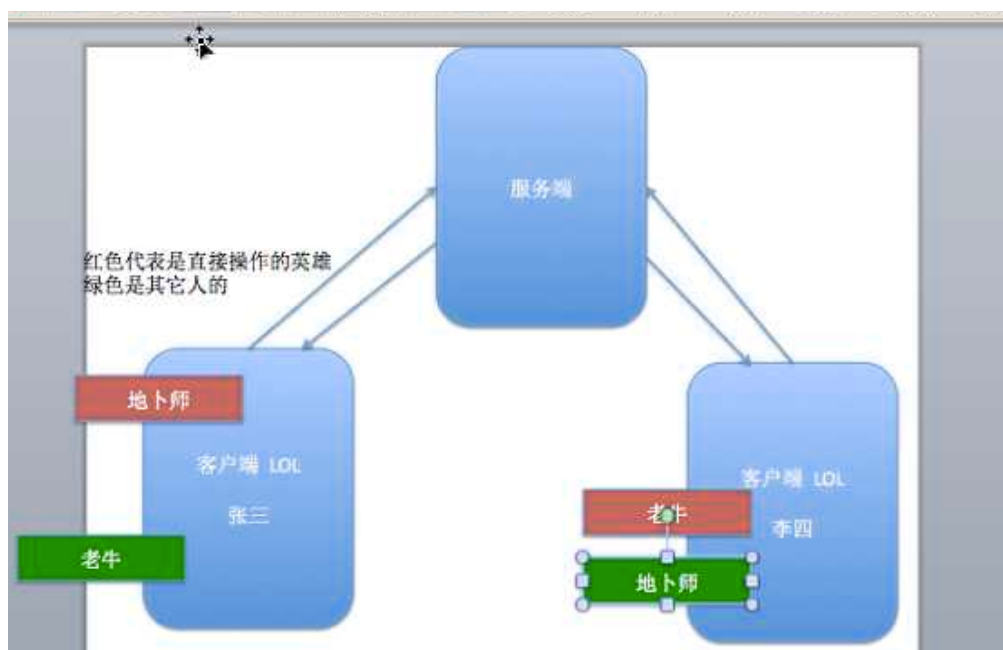
4)TCP/UDP使用场景:

TCP一般用于网络聊天(长连接.相对HTTP来说,服务器的长连接数是有限的).长连接,客户端可以主动给服务端发送请求,服务端也可以主动给客户端发送请求,聊天要保证数据的安全性.



UDP: 游戏,QQ视频,红蜘蛛都是用UDP传输的

(只管发,不管收到没有)



11.http协议的实现

HTTP:是一中协议,超文本传输协议,定义了网络传输的格式.(另一个称号叫短连接).

如果利用HTTP做聊天,每次都要重新创建连接,因为HTTP是短链接.一次回话后就断开了.如果利用HTTP做聊天,如果聊天特别频繁,会不断的创建连接,消耗资源,性能不好.)服务端不会主动给客户端发送请求.

